

# Design Pattern : patrons de conception

- ❏ Décrit une solution connue, testée et éprouvée à un problème récurrent (ici de conception)
- ❏ Cette solution s'exprime sous la forme d'un agencement de classes ou interfaces (spécification en UML)

❏ Un 30 aine de DP connus (GOF).

❏ 3 catégories :

**Les patterns de création (creational patterns) :** *Abstract Factory, Builder, Factory Method, Prototype et Singleton.*

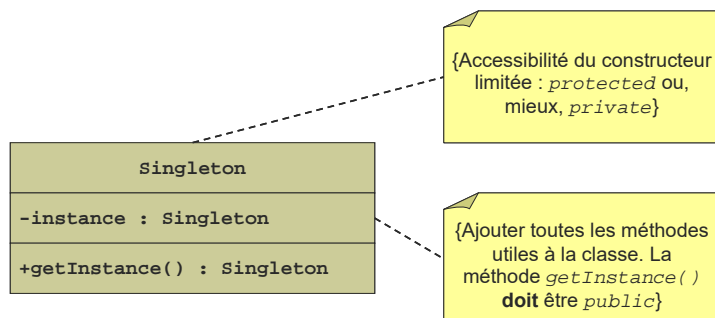
**Les patterns de structure (structural patterns) :** *Adapter, Bridge, Composite, Decorator, Facade, Flyweight et Proxy.*

**Les patterns de comportement (behavioral patterns) :** *Chain of Responsibilities, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method et Visitor.*

1

## Design Pattern : Singleton

- ❏ **Objectif :** on veut s'assurer qu'une seule et unique instance d'une classe est créée pendant toute la durée de l'application (par exemple connexion à BD).



```
public class Singleton
{
    private static Singleton instance = null;

    public static Singleton getInstance()
    {
        if (instance == null) instance = new Singleton();
        return (instance);
    }

    private Singleton() {}
}
```

2

## Singleton en environnement multi thread

```
public class Singleton
{
    private static Singleton instance = null;


    public static Singleton getInstance()
    {
        if (instance == null)           // 1
            instance = new Singleton(); // 2
        return (instance);              // 3
    }

    private Singleton() {}
}
```

1. Thread 1 calls the `getInstance()` method and determines that `instance` is null at //1.
2. Thread 1 enters the if block, but is preempted by thread 2 before executing the line at //2.
3. Thread 2 calls the `getInstance()` method and determines that `instance` is null at //1.
4. Thread 2 enters the if block and creates a new `Singleton` object and assigns the variable `instance` to this new object at //2.
5. Thread 2 returns the `Singleton` object reference at //3.
6. Thread 2 is preempted by thread 1.
7. Thread 1 starts where it left off and executes line //2 which results in another `Singleton` object being created.
8. Thread 1 returns this object at //3.

3

## Singleton en environnement multi thread

 **Solutions :** synchronisation de la création de l'instance.

### Thread-safe getInstance()

```
public class Singleton
{
    private static Singleton instance = null;

    public static synchronized Singleton getInstance()
    {
        if (instance == null)           // 1
            instance = new Singleton(); // 2
        return (instance);              // 3
    }
}
```

### Thread-safe getInstance()

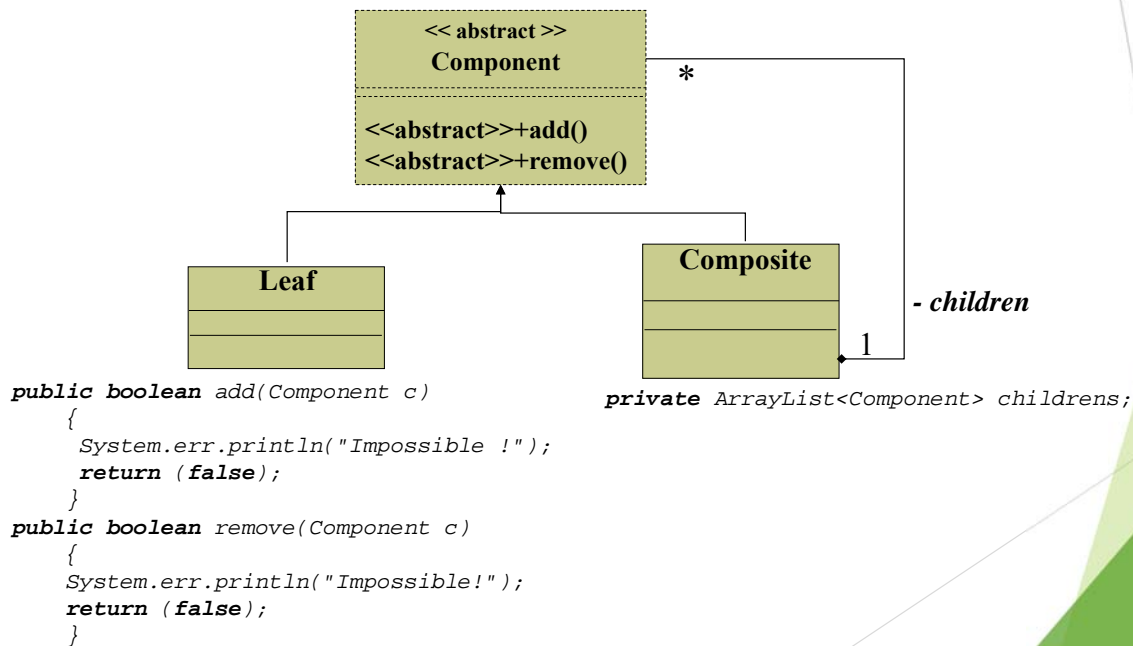
```
public class Singleton
{
    private static Singleton instance = null;

    public static Singleton getInstance()
    {
        if (instance == null) {
            synchronized(Singleton.class) {instance = new Singleton(); }
        }
        return (instance);
    }
}
```

4

## Design Pattern : Composite

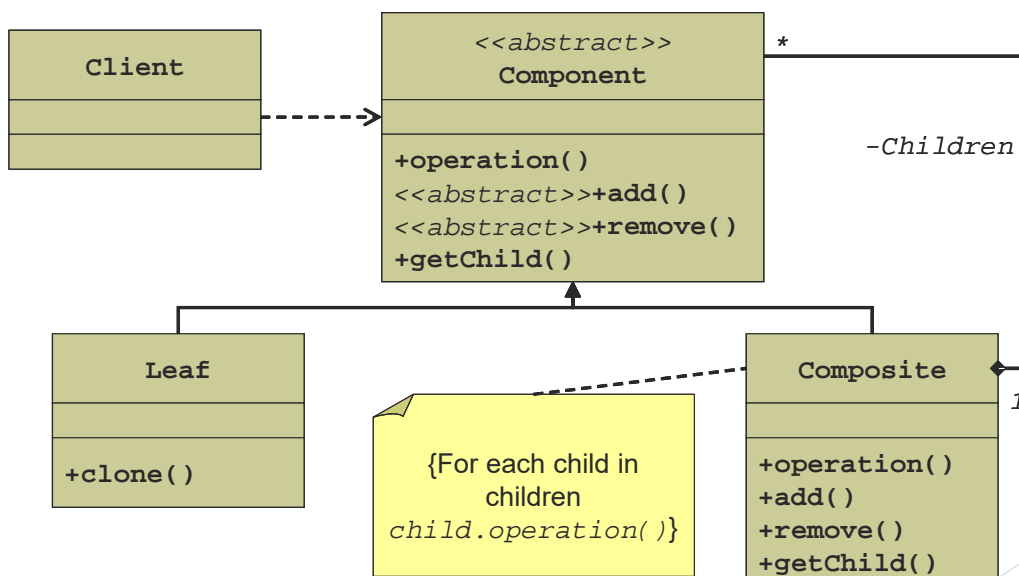
**Objectif :** représenter des hiérarchies complexes (i.e. récursives) de façon à n'avoir à distinguer que le moins possibles les nœuds et les feuilles



5

## Design Pattern : Composite

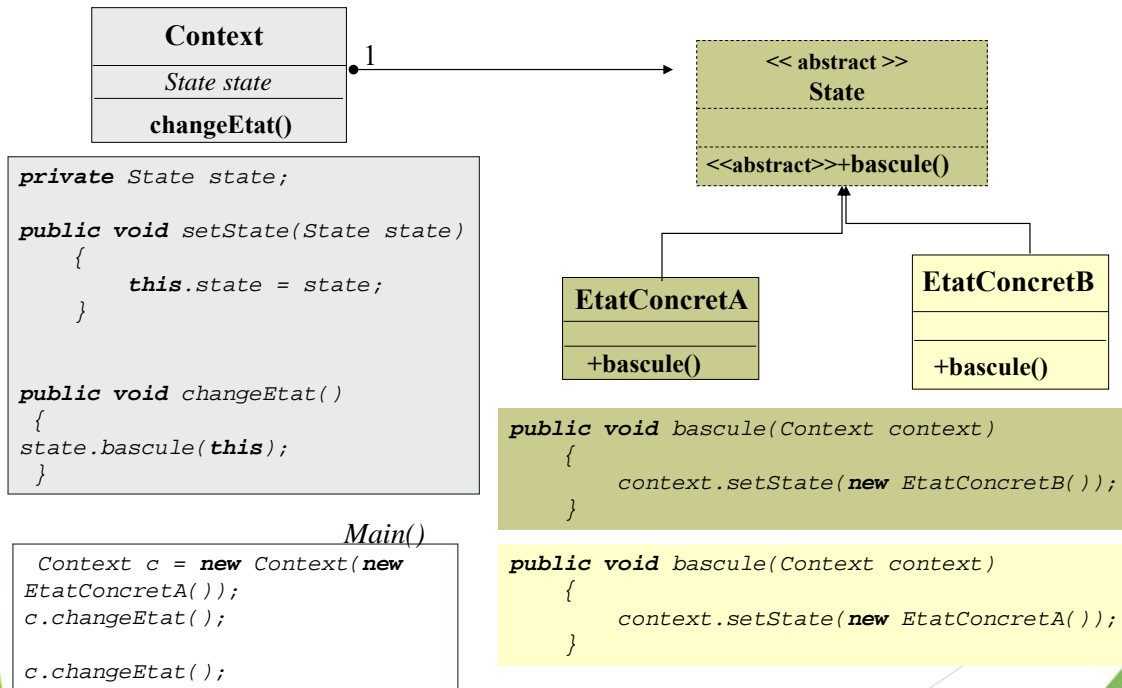
**Objectif :** des hiérarchies complexes (i.e. récursives) de façon à n'avoir à distinguer que le moins possibles les nœuds et les feuilles.



6

## Design Pattern : State

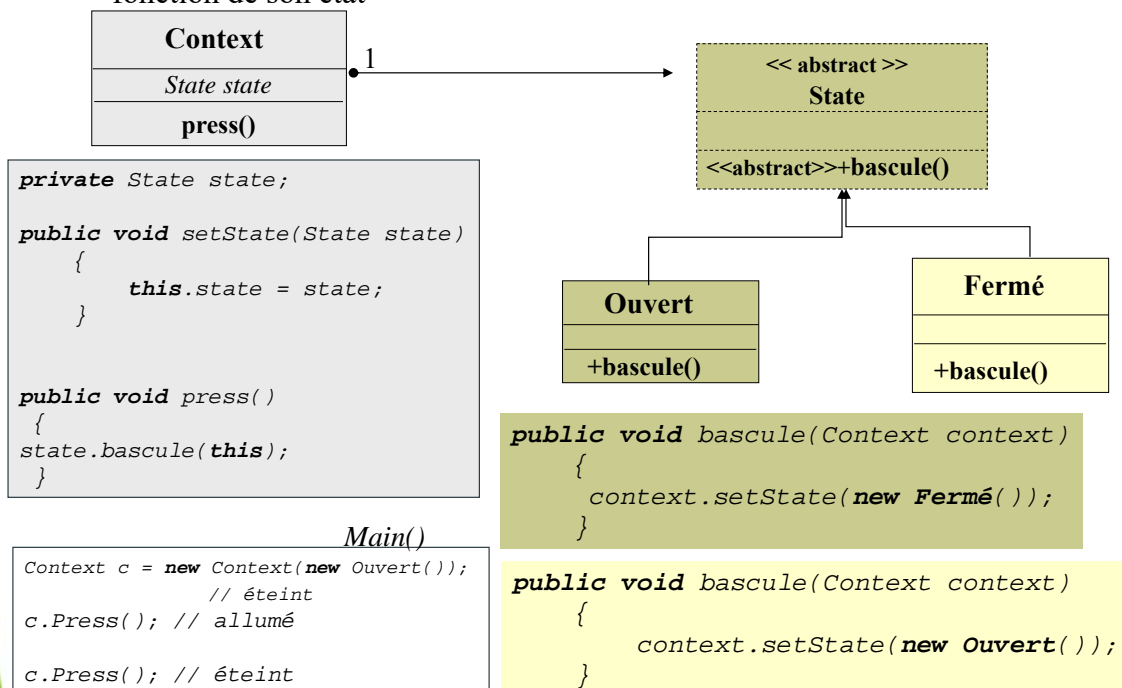
**Objectif :** ce pattern permet de faire varier le comportement d'un objet en fonction de son état



7

## Design Pattern : State

**Objectif :** ce pattern permet de faire varier le comportement d'un objet en fonction de son état



8

## Design Pattern : State Exemple

Metro

